类(一)

简单的类

```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");
```

类的继承

通过 extends 关键字 可以实现子类继承父类的属性

```
class Animal {
    move(distanceInMeters: number = 0) {
        console.log(`Animal moved ${distanceInMeters}m.`);
    }
}

class Dog extends Animal {
    bark() {
        console.log('Woof! Woof!');
    }
}

const dog = new Dog();
dog.bark();
dog.move(10);
dog.bark();
```

因为 `Dog` 继承了 `Animal` 的功能，因此我们可以创建一个 `Dog` 的实例，它能够 `bark()` 和 `move()`

来个更复杂的例子

```
class Animal {
    name: string;
    constructor(theName: string) { this.name = theName; }
    move(distanceInMeters: number = 0) {
        console.log(`${this.name} moved ${distanceInMeters}m.`);
    }
}

class Snake extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 5) {
```

```
        console.log("Slithering...");
        super.move(distanceInMeters);
    }
}

class Horse extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 45) {
        console.log("Galloping...");
        super.move(distanceInMeters);
    }
}

let sam = new Snake("Sammy the Python");
let tom: Animal = new Horse("Tommy the Palomino");

sam.move();
tom.move(34);
```

这一次，我们使用 `extends` 关键字创建了 `Animal` 的两个子类： `Horse` 和 `Snake`

与前一个例子的不同点是，派生类包含了一个构造函数，它 *必须* 调用 `super()`，它会执行基类的构造函数。 而且，在构造函数里访问 `this` 的属性之前，我们 *一定* 要调用 `super()`

`Snake` 类和 `Horse` 类都创建了 `move` 方法，它们重写了从 `Animal` 继承来的 `move` 方法，使得 `move` 方法根据不同的类而具有不同的功能。 注意，即使 `tom` 被声明为 `Animal` 类型，但因为它的值是 `Horse`，调用 `tom.move(34)` 时，它会调用 `Horse` 里重写的方法